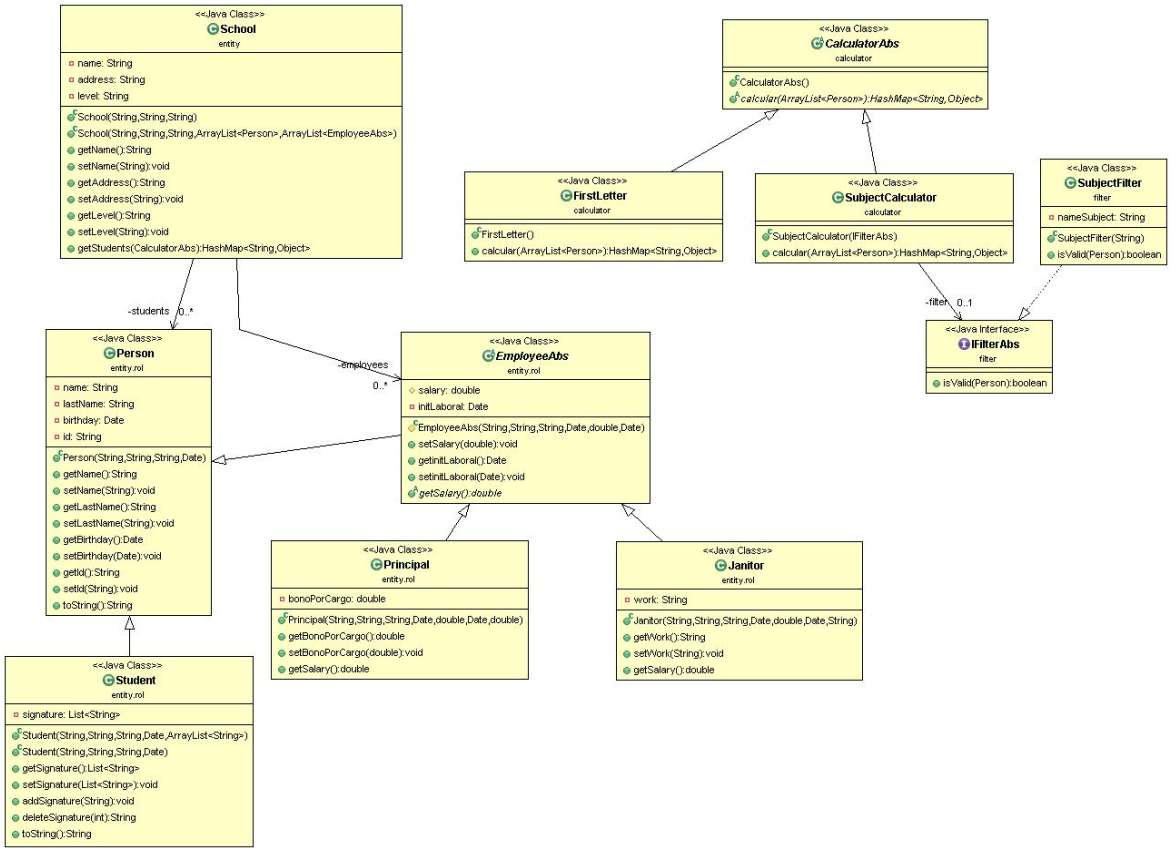
Ejercicio N° 1

A. Se adjunta la imagen del diagrama de Clases UML correspondiente a la solución para el ejercicio N° 1.



B. En el proyecto ***TestTeraCode*** se encuentra la solución implementada en Java para este punto.

C. En el proyecto ***TestTeraCode*** se encuentra la solución implementada en Java para este punto.

D. Como se puede observar en el ***Diagrama 1*** el punto fuerte del esquema es la distribución de los datos, la misma se encuentra realizada de tal forma que no hay redundancia de datos (Normalizada). Mediante esta distribución también se logra optimizar la actualización de los datos, dado que al realizar una modificación esta no repercute en varias tablas. Otro punto fuerte es que se puede tener índices sin elevar demasiado los costos de actualización.

La contra de este esquema es los costos de consulta de datos, dado que se puede llegar a necesitar realizar múltiples joins para obtenerlos. Las consultas deberán ser optimizadas correctamente

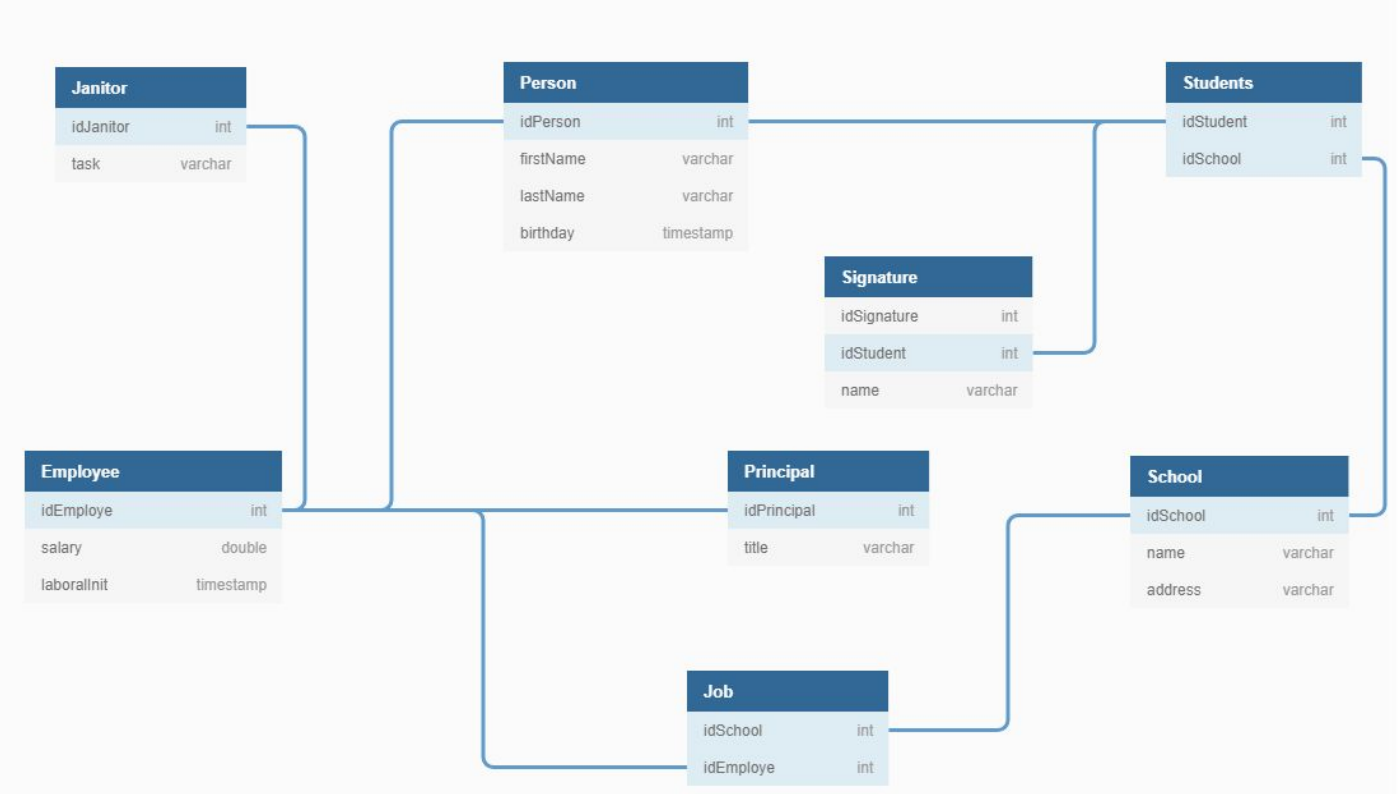


Diagrama 1

En cambio en el ***diagrama 2*** lo que podemos observar que lo que se tuvo en cuenta fue no tener la necesidad de realizar tantos joins para obtener los datos deseados (se aumenta la redundancia de datos, el esquema no está normalizado) esto nos produce un aumento de tamaño en la base de datos, y un mayor costo para mantenerla actualizada.

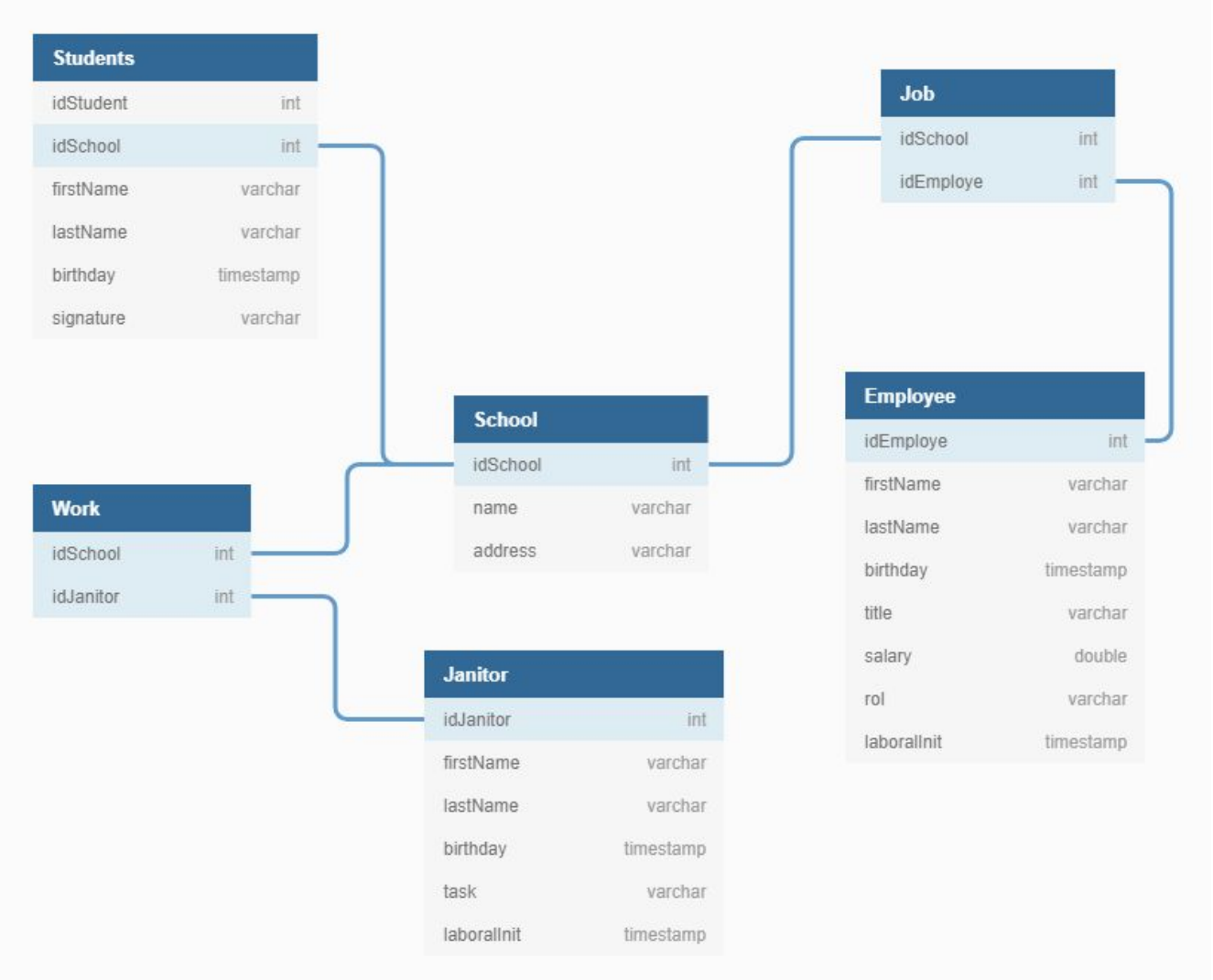


Diagrama 2

E. Existen varias optimizaciones que se pueden realizar a la consulta en cuestión.
   a. En primer lugar se está trayendo todos los atributos cuando solo se necesitan el nombre y apellido del ***JANITOR.***
   b. Otro punto a mejorar es la cantidad de joins que se realizan y el porqué se realizan, traer los datos de la tabla ***JANITOR y EMPLOYEE***

no es necesario dado que los datos nombre y apellido ya los tenemos disponibles en la tabla ***PERSON***. Para mejorar esto lo que se puede hacer es enviar estos joins a una subconsulta para validar que el id perteneciste a la tabla ***PERSON*** se encuentre en las tablas ***JANITOR y EMPLOYEE***. Esto nos trae como ventaja no estar acarreando tantos datos desde un inicio por la unión de tablas.

Ejemplo de consulta obtenida:

```
SELECT P.name, P.lastName
FROM PERSON P
WHERE EXISTS (
        SELECT 1
        FROM EMPLOYEE E INNER JOIN JANITOR J ON E.id = J.id
        WHERE J.workingArea = 'Hallaway' AND P.id = J.id
)
```

c. Por último se podría agregar indices por los campos a obtener de la búsqueda, de esta forma se podría acelerar los tiempo respuesta.

F. Al ser tablas las cuales no son actualizadas con frecuencia, una de las posibles mejoras a realizar es la implementación de índices. Al no se ser actualizadas con frecuencias el índice se verá afectado en menor medida por lo cual puede ser una solución viable.

El tipo de índice a utilizar va a variar dependiendo del tipo de dato que retorna la consulta, pero si el mismo busca devolver datos por rango podría utilizarse un ARBOL B+ (Buenos para resolver este tipo de búsquedas).

Otro tema a ver es la cantidad de join que se utilizan en la consulta, dado que como vimos en el punto anterior tal vez mediante la restructuración de la misma pueda ser mejorada. Se puede llegar a disminuir la cantidad de joins utilizados, implementar subconsultas.

G. La query que se solicita tiene ciertas similitudes con la consulta del punto E. Para la misma los datos nombre y apellido se encuentran disponibles en la tabla Person por lo tanto solo necesitamos validar que efectivamente esa row de Person machea con una row en la tabla Student. Para este fin la forma más eficiente es la utilización de una subconsulta y que esta se encuentre en un EXISTS para que ante la primera condición true se prosiga con la siguiente row de Person.

```
SELECT P.name, P.lastName
FROM PERSON P
WHERE EXISTS (
            SELECT 1
            FROM Student S
            WHERE S.id = P.id
      )
AND DATEDIFF(day,birthdate,GETDATE())/365 BETWEEN 19 AND 24
```

H. La forma en que se puede llevar a cabo la implementación de la lógica de negocio en la Base de Datos es mediante la implementación de ***Stores Procedures.*** A su vez se necesitaría la implementación de ***Triggers y Checks*** para controlar todas la validación y reglas a la hora de realizar las operaciones de Agregar-Actualizar-Borrar datos sobre las tablas.

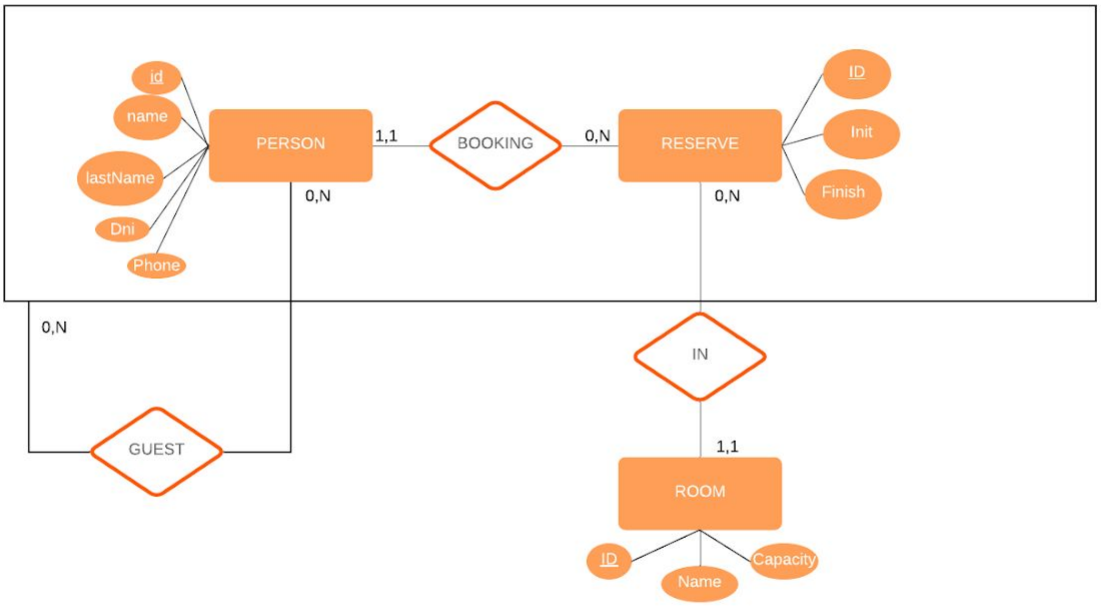
Se implementará índices para aumentar los tiempos de accesos a los datos de mayor necesidad siempre teniendo en cuenta la tasa de actualización de datos de dichas tablas. Dado que si son actualizadas con mucha frecuencia esto podría ser contraproducente (Se pasaría mucho tiempo actualizando índices, repercutiendo en la performance).

Pros y Contras: Esta solución no tiene puntos buenos, debido a que estamos dando toda la responsabilidad a la base de datos, a su vez la complejidad de mantenimiento aumenta.

Lo mejor seria darle a la base de datos solo la responsabilidad del acceso y guardado de datos. Mientras que en una capa de servicio se podría realizar la lógica de negocio.

Ejercicio N° 2

A. A continuación se muestra el diagrama de Entidades y Relaciones con las entidades básicas para poder hacer una reservación.

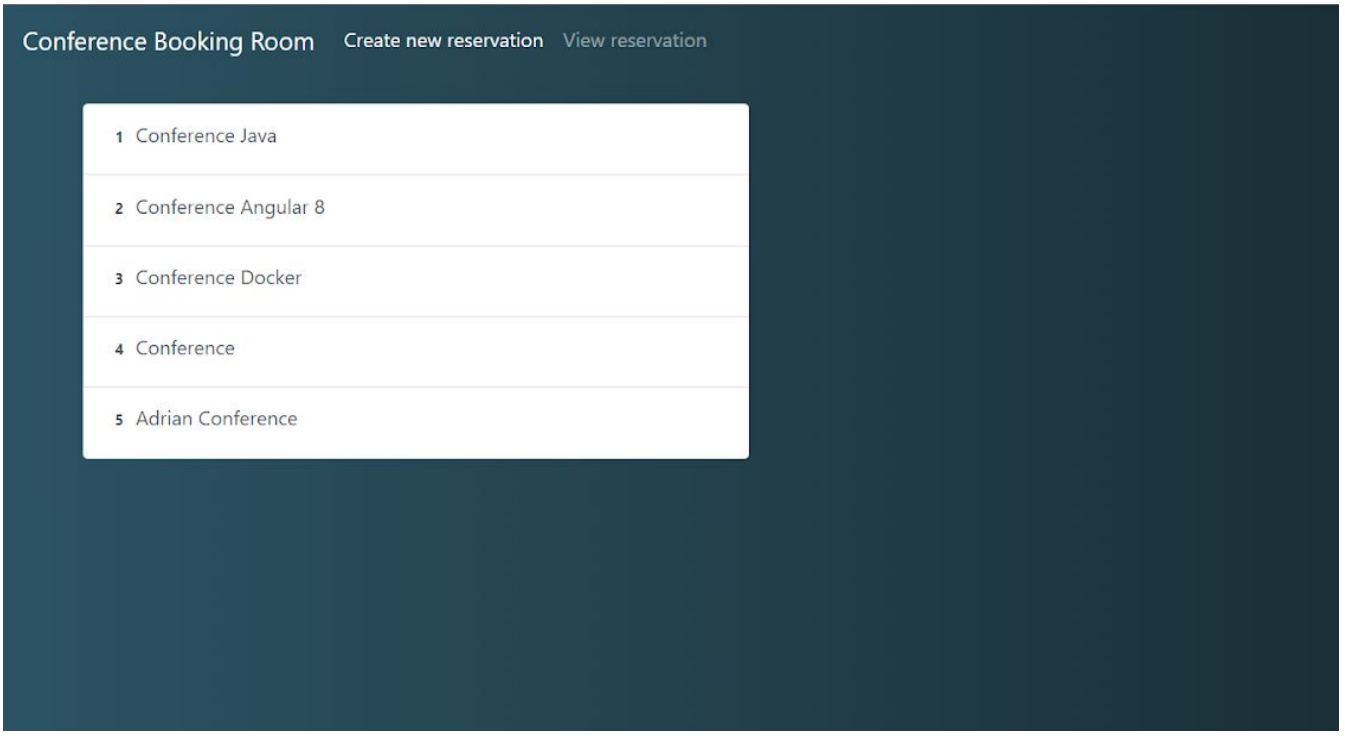


B. Se adjuntan capturas de pantalla con el diseño de las interfaces para la carga y vista de las reservas realizadas.

La misma posee una barra de navegación desde la cual se puede ir a la creación de una nueva reserva o si se selecciona la otra opción se puede ver las reservas ya realizadas.

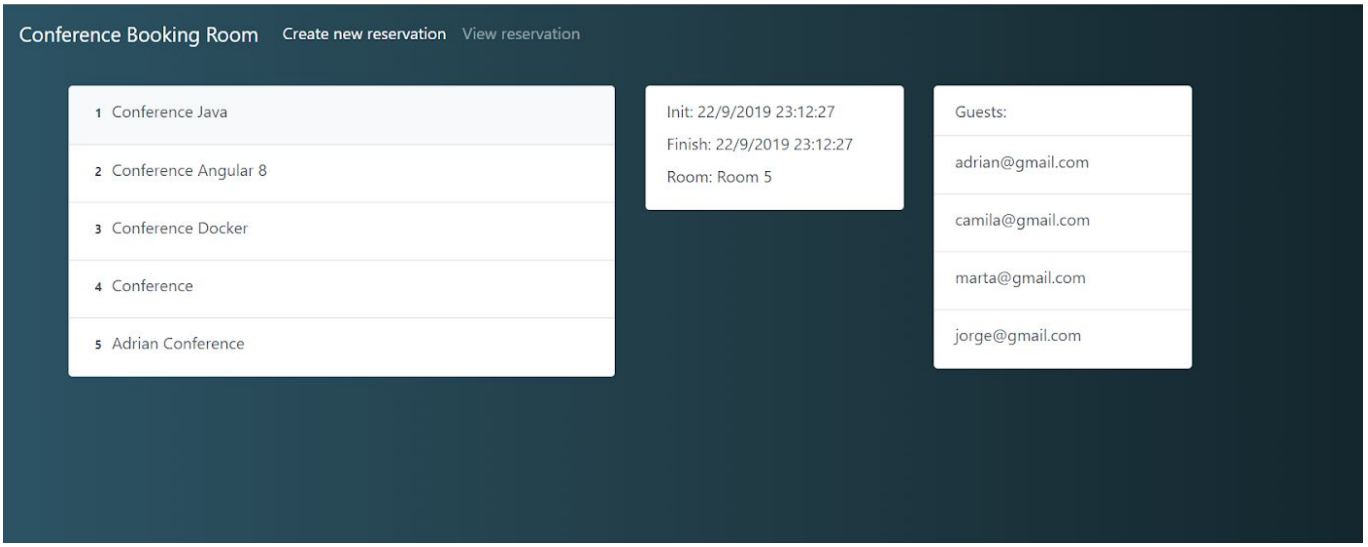
View Reservation

En la siguiente captura se puede ver la interface que permite visualizar las “Conference” creadas.



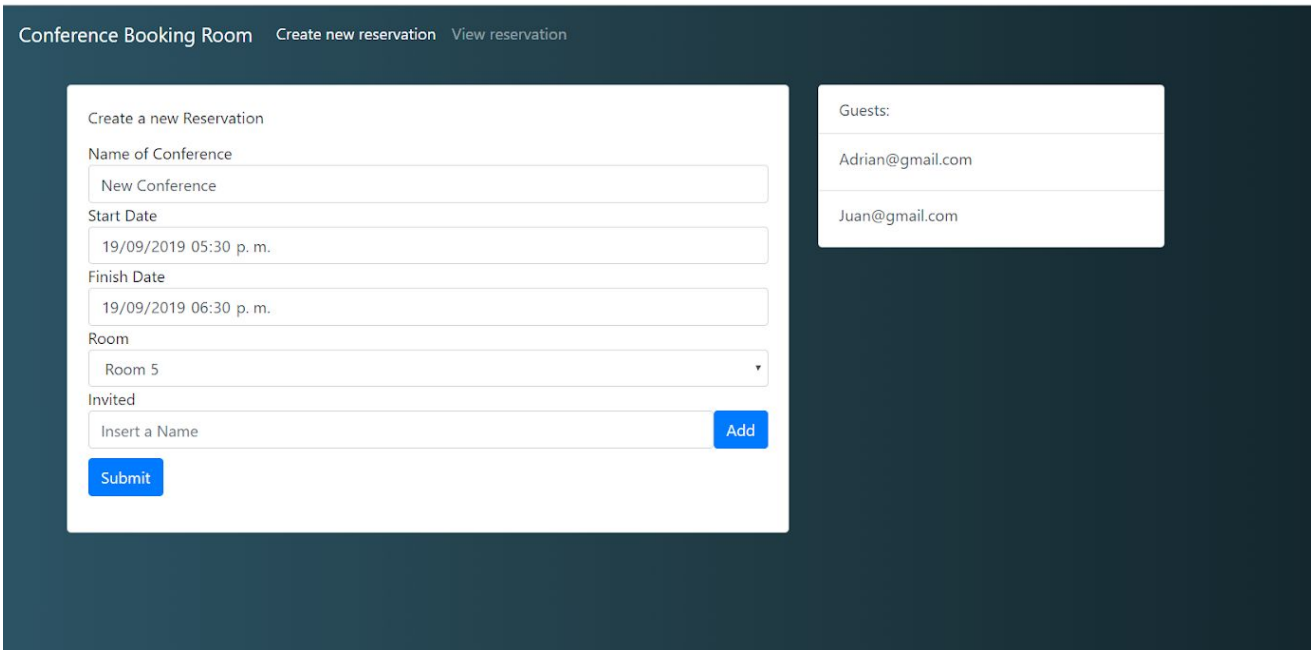
View Reservation Detail

Se muestra a continuación los desplegables que habilitan cuando se da click a una de las Conference



Create a new reservation

En la siguiente interfaz se define la creación de una nueva “Conference” en la misma se permite la entrada del nombre, día y hora de inicio y fin, como así también la ca5rga del cuarto y la los invitado.
La carga de los invitados va completando la lista que se visualiza en la parte izquierda de la imagen.



C. Antes de realizar el insert en la base de datos se ejecutar el siguiente Trigger.

```
CREATE TRIGGER validBooking
BEFORE INSERT ON RESERVE FOR EACH ROW
BEGIN
  DECLARE @MIN INT
  SET @MIN = DATEDIFF(MINUTE, NEW.INIT, NEW.FINISH)
  IF @MIN > 15 AND @MIN <180 THEN
          IF NOT EXISTS ( SELECT 1 FROM RESERVE R
                                  WHERE ID= NEW.ID
                                   AND (
                                     NEW.INIT BETWEEN R.INIT AND R.FINISH
                                     OR NEW.FINISH BETWEEN R.INIT AND R.FINISH
                                   )
                             ) THEN
                  INSERT INTO RESERVE (INIT, FINISH, IDPERSON, ID)
                  VALUE (NEW.INIT, NEW.FINISH, NEW.IDPERSON, NEW.ID)
          ENDIF;
  ENDIF;
END;
```